Available online at www.sciencedirect.com

ScienceDirect

Discrete Applied Mathematics 156 (2008) 556–568

DISCRETE APPLIED MATHEMATICS

www.elsevier.com/locate/dam

Batch processing with interval graph compatibilities between tasks

Gerd Finke, Vincent Jost, Maurice Queyranne, András Sebő

Laboratoire Leibniz-IMAG, Grenoble, France

Received 4 October 2004; received in revised form 1 March 2006; accepted 3 March 2006
Available online 19 April 2007

Abstract

We analyze batch-scheduling problems that arise in connection with certain industrial applications. The models concern processing on a single max-batch machine with the additional feature that the tasks of the same batch have to be compatible. Compatibility is a symmetric binary relation—the compatible pairs are described with an undirected "compatibility graph", which is often an interval graph according to some natural practical conditions that we present. We consider several models with varying batch capacities, processing times or compatibility graphs. We summarize known results, and present a min–max formula and polynomial time algorithms.

Keywords: Batch-scheduling; Task compatibilities; Interval graphs; Perfect graphs; Bounded coloring; Max-coloring

1. Introduction

A *batch machine*refers to a machine that can process several tasks simultaneously. We consider here the so-called *max-batch* or *parallel-batch* (sometimes abbreviated *p-batch*) machine, where the processing time of a group of tasks, called a *batch*, is the longest processing time of the tasks it contains. The initial motivation for this branch of scheduling theory was the scheduling of semiconductor burn-in operations [23]. Intensive research has subsequently been developed on this subject for various scheduling objectives and additional constraints, see for instance the surveys [9,29].

In this paper we focus on minimizing the makespan $C_{\max}$. In addition, we assume that the tasks in the same batch have to be *compatible*, for instance they must share similar physical properties (form, weight, etc.). The problem that we want to analyze may be formulated as follows. There are n independent tasks T_j $(j = 1, \ldots, n)$ to be scheduled on a single max-batch machine. The batch machine has *capacity* b, which means that at most b tasks can be processed simultaneously (b may be finite or infinite). Each task T_j has a (minimal) processing time p_j. A batch B has processing time $p(B) = \max\{p_j : T_j \in B\}$ and all tasks in the same batch start and finish at the same time. Preemption is not allowed. Tasks in the same batch have to be *pairwise compatible*. This relation is represented by a *compatibility graph* $G = (V, E)$, where V is the set of tasks and a pair of tasks is an element of the edge set E if and only if they are compatible. By definition, a batch forms a *clique* (a complete subgraph, not necessarily maximal) in the compatibility graph G. Since all tasks have to be executed, the problem is to find a decomposition of G into cliques $B_1, B_2, \ldots, B_l$, where l is not known in advance, such that the schedule length $C_{\max} = \sum_i p(B_i)$ is minimized. These batches may then be scheduled in any order, without any idle time.

E-mail addresses: gerd.finke@imag.fr (G. Finke), vincent.jost@imag.fr (V. Jost), maurice.queyranne@imag.fr (M. Queyranne), andras.sebo@imag.fr (A. Sebő).

0166-218X/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2006.03.039

The concept of scheduling with task compatibilities has been treated in [3–7] for general graphs and also for some special graphs. This theory is related to chromatic scheduling [11] where the complementary graph (graph of incompatibilities) is considered, leading to a graph coloring problem. For chromatic scheduling, there are usually no capacity constraints.

We describe in Section 2 two specific industrial applications. They illustrate that interval graphs occur quite naturally as compatibility graphs in batch processing. Recall [19,20] that an *interval graph* $G = (\mathcal{I}, E)$ is a graph for which the node set $\mathcal{I} = \{I_1, \ldots, I_n\}$ can be identified with a set of intervals on the real line, such that two nodes $I_1, I_2 \in \mathcal{I}$ are adjacent in G if and only if the intervals intersect (that is, $I_1 \cap I_2 \neq \emptyset$). In Section 3, the batch-scheduling models are formulated, whereas the necessary terminology from graph theory can be found in Section 4. Solution methods are presented in Section 5. In the final Section 6, the inclusion of release dates is discussed and some final remarks are presented.

2. Industrial applications

Application I : In [26] a rolling-mill is described where the metal goes through repeated cycles of rolling and heating to produce the final steel plates. In the heating phase, the metal in form of coils is piled up on a base and then heated together in a bell furnace. Each coil T_j has to be heated for at least p_j time units. A loading of the furnace represents a max-batch. The material heated together has to be compatible, which means in this case that the coils have to have similar heights. One may express these compatibilities by means of a tolerance height Δ and assign to a coil of height H the tolerance interval $[H - \Delta/2, H + \Delta/2]$. Two metal coils are compatible if their tolerance intervals intersect. For a given set of metal coils (tasks), one obtains an interval graph for the task compatibilities, and a clique in this graph defines an admissible batch for the furnace. The batch size is limited by a given finite capacity b. The objective is to minimize the total heating time $C_{\max}$.

Application II : This industrial application has been studied in [8,17,18]. A manufacturer is developing flow-lines for the production of metallic office equipment. In a first phase, a large number of holes of various shapes and sizes are to be punched into a metal sheet which is then bent in a second operation. The essential features of the hole-punching facility are displayed in Fig. 1. Several heads are arranged in two parallel lines, each one equipped with a tool magazine, and the metal sheet is moving unidirectionally through the system. The heads can move perpendicular to the direction of the metal sheet and also have a small lateral movement of size Δ. Each hole-punching operation (of one or several holes) requires a stopover and positioning of the metal sheet in the system. During the repositioning of the metal sheet, each active head is moved to its hole-punching position and the tool magazine rotates to place the required tool into its working position. There is, therefore, no additional tool changing time. During a stopover, all the holes that can be reached by the various heads (equipped with the appropriate tool) are grouped together in a work phase

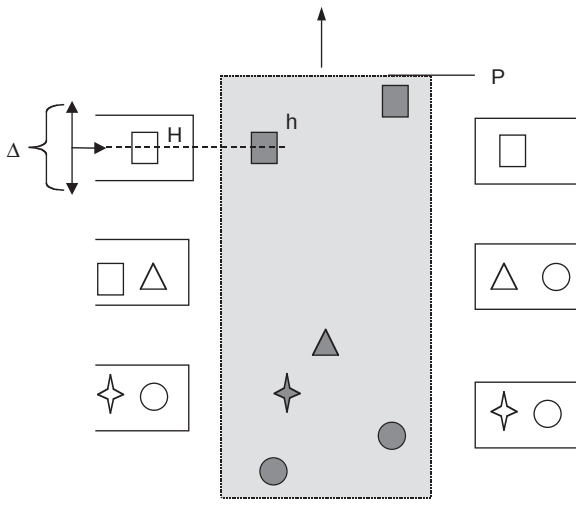

Fig. 1. A manufacturing system with six hole-punching heads.

(forming a batch) and punched simultaneously. The batches are again max-batches since the longest hole-punching operation defines the duration of the work phase.

In this application, the task compatibilities are defined by the geometry of the system. We may define the compatibility graph $G = (V, E)$ as follows. A node of V is a pair (h, H), where h is a hole and H is a head whose tool magazine contains the suitable tool for h and is situated on the appropriate side of the two-line system. It is assumed that for each hole h, exactly one head H has been pre-assigned. With each node (h, H) we associate the topmost position P of the metal sheet for which h is in front of the central position of H. Then hole h can be made by head H for all positions of the metal sheet in interval $I(h, H) = [P - \Delta/2, P + \Delta/2]$. If two such intervals $I(h, H)$ and $I(h', H')$ intersect, then the two corresponding holes $h \neq h'$ can be punched in one work phase, provided that $H \neq H'$. Because of these restrictions ($H \neq H'$), the compatibility graph is only "almost" an interval graph. Additional features of this application are described in the next section. The objective is to minimize the total hole-punching time $C_{\max}$. For the solution approach in [8,17,18], the missing edges for obtaining an interval graph are taken care of and the problem is solved approximately for all possible assignments of heads to holes. In the following, we will simplify the study and will mostly assume that G is an interval graph but we generalize the model with respect to the other parameters.

3. Batch-scheduling models

We consider batch-scheduling problems on a single max-batch machine with task compatibilities. We use the notations of [10]:

$$1/p\text{-}batch, G = \beta_1, \beta_2/C_{\max},$$

where the compatibility graph is specified by the parameter β_1; we use $\beta_1 = INT$ for an interval graph and $\beta_1 = (V, E)$ for a general graph. The parameter β_2 specifies the batch capacity b as follows: "$b = k$" for a fixed capacity; "$b < n$" for variable finite capacity b which is part of the input; or β_2 is void for infinite capacity. There may be other parameters β_i representing additional restrictions, for instance structured processing times, release dates, etc.

Referring to Application II, one may as a first approximation consider the processing times as being constant ($p_j = 1$). The size of the maximal cliques in the compatibility graph $G = INT$ cannot exceed the number of hole-punching heads, so that there is no limitation of the batch capacity. This leads to the problem

$$\text{(P1)} \quad 1/p\text{-}batch, G = INT, \quad p_j = 1/C_{\max}.$$

Problem (P1) has been solved approximately in [8,17,18] for graphs that are slightly more general than interval graphs.

A more careful analysis of the model yields the following. For each positioning of the metal sheet in the system, a certain number of heads will be active (punching holes). For the process to work properly, at least one head on each side should be inactive since they are required to hold the metal sheet in position. Therefore, the more precise model would be

$$\text{(P2)} \quad 1/p\text{-}batch, G = INT, \quad b < n, \quad p_j = 1/C_{\max}.$$

In Application II, b is the total number of heads minus two.

On the other hand, the hole-punching durations may vary. In fact, holes with large diameters have to be cut in several successive operations along the contours of the circle (or ellipsoid). This gives two more models with arbitrary processing times:

$$\text{(P3)} \quad 1/p\text{-}batch, G = INT/C_{\max},$$

$$\text{(P4)} \quad 1/p\text{-}batch, G = INT, \quad b < n/C_{\max}.$$

4. Definitions and notations from graph theory

Problems (P1)–(P4) can be described in the language of graph theory. For $U \subseteq V$, let $G(U) := (U, E(U))$ be the subgraph induced by U and $G - U := G(V \backslash U)$ be the graph induced by vertices not in U. A *clique* in a graph $G = (V, E)$ is a set of nodes $U \subseteq V$ inducing a complete subgraph. The *clique number* $\omega(G)$ of a graph G is the maximum number of vertices inducing a clique in G. The complementary graph $\overline{G} = (V, \overline{E})$ of $G = (V, E)$ is defined

by $(a, b) \in E(\overline{G})$ if and only if $(a, b) \notin E(G)$. A *stable set* in a graph G is clique in $\overline{G}$ and $\alpha(G) = \omega(\overline{G})$. The minimum number of stable sets needed to partition V is the *chromatic number* $\chi(G)$ of G. We denote the minimum number of cliques needed to partition V by $\overline{\chi}(G)$. One has $\overline{\chi}(G) = \chi(\overline{G})$. A *matching* in a graph is a collection of disjoint edges. An *edge cover* is a collection of edges covering all the vertices. A vertex is *lonely* if it is has no neighbor.

(P1)–(P4) ask for a partition of the node set of G into cliques $B_1, \ldots, B_\ell$ (whose number ℓ is not specified in advance). These cliques need not be maximal and if one allows an overlapping of the cliques, then its contribution to the objective would increase. Therefore, the problems are also equivalent to find the minimum number of cliques that cover the node set (that is to find a family of possibly overlapping cliques whose union is V). The objective is to minimize the number of cliques in problems (P1) and (P2), and the total processing time $\sum_i p(B_i)$ of all cliques in problems (P3) and (P4). In addition, problems (P2) and (P4) constrain the size $|B_i|$ of each clique not to exceed b.

We assume that an interval graph $G = (V, E)$ is given by corresponding intervals $V = \mathscr{I} = \{I_1, \ldots, I_n\}$ where $I_i = [a_i, b_i]$ $(i = 1, \ldots, n)$, sorted in a nondecreasing order of the *terminal endpoints* b_i. The other endpoint a_i is the *initial* endpoint of I_i. (Practice provides the intervals themselves, and interval graphs can be recognized and reconstructed in linear time $O(|V| + |E|)$ [19]). From now on we identify V and $\mathscr{I}$ and interchangeably use the words *task*, *interval* and *node*. A *simplicial vertex* of a graph is a vertex whose neighbors form a clique. $v_1, \ldots, v_n$ is a *simplicial order* if v_i is a simplicial vertex in $G_i := G(v_i, \ldots, v_n)$. Any nondecreasing order of terminal endpoints in an interval graph is a simplicial order. *Chordal graphs* are those that have a simplicial order (an equivalent definition [19] is that chordal graphs do not have chordless circuits of length greater or equal to 4). G is a *split graph* if its vertex set can be partitioned into two sets, one inducing a clique and the other inducing a stable set. G is a split graph if and only if both G and $\overline{G}$ are chordal [19]. A graph is *circular arc* if it is the intersection graph of a family of intervals drawn on a circle. A *Helly-clique* in a circular-arc graph is a set of arcs which share a common point on the circle. Helly-cliques are cliques of the intersection graph, but not vice-versa. A graph is P_4*-free* if it has no induced path on four vertices.

5. Solution methods

5.1. Problem (P1)

This problem is equivalent to covering the nodes of the compatibility graph with the smallest number of cliques. The problem is solved in polynomial time for interval graphs and the more general class of "perfect graphs", as well as for other graph classes (for instance circular-arc graphs), whereas it is $\mathscr{NP}$-hard for general graphs, see [19,20,30].

For illustration, we solve problem (P1) for the interval graph in Fig. 2. The solution is found in a greedy fashion: the first batch B_1 is the largest batch containing the first interval $I_1 = [a_1, b_1]$, that is, B_1 consists of all intervals containing b_1. This greedy principle is then recursively applied to the remaining intervals. Starting with the first interval I_1, one gets the largest possible clique $B_1 = \{I_1, I_2, I_3, I_4, I_5, I_7\}$ containing I_1. Continuing with the next remaining interval,

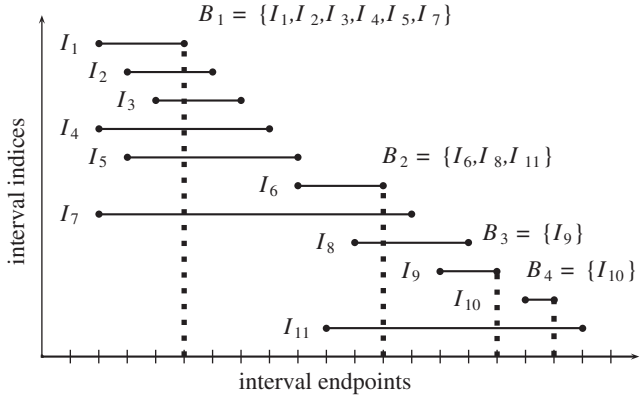

Fig. 2. A list of intervals given by nondecreasing finishing time, with the optimal solution to (P1) from the greedy algorithm.

I_6, yields the second clique $B_2 = \{I_6, I_8, I_{11}\}$, then comes $B_3 = \{I_9\}$ and finally $B_4 = \{I_{10}\}$. Hence an optimal schedule is of length $C_{\max} = 4$.

5.2. Problem (P2)

In problem (P2) we want to minimize the number of cliques in the cover, subject to the restriction that every clique must have no more than b nodes. For $b = 2$ this problem is solvable in polynomial time for general compatibility graphs [7], since it reduces to finding a maximum cardinality matching in G. (P2) is $\mathcal{NP}$-hard in P_4-free graphs, as well as in complementary of bipartite graphs [1]. For any fixed $b \geqslant 4$, (P2) is $\mathcal{NP}$-hard in complementary of interval graphs [1] and for any fixed $b \geqslant 6$, it is $\mathcal{NP}$-hard in permutation graphs [24]. In contrast, we provide a simple greedy algorithm to solve (P2) in interval graphs. We realized after submission that a solution has already been published in [1] using earlier results [27]. Our original algorithm is kept for pedagogical reasons since we improve on earlier results by providing a min–max relation for (P2). This relation holds for interval graphs and in several other interesting cases. (P2) is also polynomial in split graphs [1,3,16] and even in chordal graphs [25]. In [21,1], the so-called *bounded coloring* problem (equivalent to (P2) by taking the complementary graph) is studied and solved for some other classes of graphs (see [16] for a survey).

A *b-clique* in a graph G is a clique Q of G with size $|Q| \leqslant b$. A *b-clique cover* of the graph $G = (V, E)$ is a family $\mathcal{B}$ of b-cliques that cover V. Its *size* is $|\mathcal{B}|$. Letting $B(G, b)$ denote the set of all b-clique covers of G, the optimum objective value of problem (P2) is

$$\text{(P2)} \quad \overline{\chi}_b(G) := \min_{\mathcal{B} \in B(G,b)} |\mathcal{B}|.$$

Note that $\overline{\chi}_b(G)$ is a *monotonous* function of G, that is, $\overline{\chi}_b(G) \geqslant \overline{\chi}_b(G - v)$ for all $v \in V(G)$.

For $U \subseteq V$, let $C_1(U), C_2(U), \ldots, C_t(U)$ be the node sets of the connected components of $G(U)$ and

$$\sigma_b(G, U) := \sum_{i=1}^{t} \left\lceil \frac{|C_i(U)|}{b} \right\rceil, \tag{1}$$

where $\lceil x \rceil$ denotes the (real) number x rounded *up* to the nearest integer. Let

$$\sigma_b(G) := \max_{U \subseteq V} \sigma_b(G, U). \tag{2}$$

Min–max relation and algorithms for $b = 2$ or $\omega(G) \leqslant 3$ or $\omega(G) \leqslant b$: If we are covering with cliques of small size or of unrestricted size, then (P2) is linked with some well-known problems.

The case $b = 2$ is equivalent to find a minimum edge cover of G:

Theorem 1. *For every graph G, one has $\overline{\chi}_2(G) = \sigma_2(G)$, or equivalently*:

$$\min_{\mathcal{B} \in B(G,2)} |\mathcal{B}| = \max_{U \subseteq V} \sigma_2(G, U). \tag{3}$$

Moreover $\overline{\chi}_2(G)$ can be computed in polynomial time.

Proof. If $v \in V$ is a lonely vertex of G, then $\overline{\chi}_2(G - v) = \overline{\chi}(G) - 1$ and $\sigma_2(G - v) = \sigma_2(G) - 1$. If G has no lonely vertices, a partition by cliques of size at most 2 is equivalent to an edge cover. The minimum in (3) is therefore equivalent to a minimum edge cover of G. (3) is therefore equivalent to [30, (27.3), p. 461]. Finding a minimum edge cover can be done in polynomial time by a matching technique [30]. □

Since it is NP-complete to decide whether a graph can be vertex-partitioned into triangles [15], which is equivalent to $\overline{\chi}_3(G) = n/3$, we cannot expect a polynomial algorithm and formula to be valid for $b = 3$. Moreover, if the capacity is not bounded, we have to deal with the ordinary coloring problem in the complementary graph which is also NP-hard [15]. We therefore restrict ourselves to graphs for which the clique partitioning problem is tractable: the graphs we will handle are perfect graphs. For $b = 3$, (P2) can be solved for perfect graphs with clique number at most 3 [21]. We prove that the min–max formula also holds:

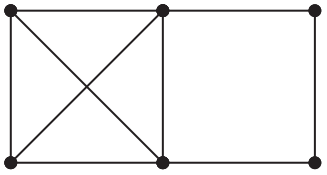

Fig. 3. Graph G such that $3 = \min_{\mathscr{B}\in B(G,3)}|\mathscr{B}| > \max_{U\subseteq V}\sigma_3(G,U) = 2$.

Theorem 2. *For every perfect graph G with $\omega(G)\leqslant 3$ and any positive integer b, one has $\overline{\chi}_b(G) = \sigma_b(G)$, or equivalently*:

$$\min_{\mathscr{B}\in B(G,b)} |\mathscr{B}| = \max_{U\subseteq V} \sigma_b(G,U). \tag{4}$$

Proof. *Case* (1): $b\leqslant 2$. The assertion follows from Theorem 1.

Case (2): $b\geqslant\omega(G)$. In this case, every clique of G is also a b-clique, so $\overline{\chi}_b(G)=\overline{\chi}(G)$. Since G is perfect, $\overline{\chi}(G)=\alpha(G)$. Noting that for any stable set S one has $\sigma_b(G,S) = |S|$, and choosing S^* to be a maximum stable set we get:

$$\overline{\chi}_b(G) = \overline{\chi}(G) = \alpha(G) = |S^*| = \sigma_b(G,S^*)\leqslant\sigma_b(G).$$

Equality holds throughout using weak duality (see (5) below). □

In case (2) above, optimal solutions can be computed in polynomial time in the framework of perfect graphs [20,30]. In the case of unrestricted capacities, we also have $\sigma_b(G) = \alpha(G)\leqslant\overline{\chi}(G) = \overline{\chi}_b(G)$, and hence equality in the case of perfect graphs:

Theorem 3. *Eq.* (4) *holds for any perfect graph G and any integer b such that $\omega(G)\leqslant b$.*

Proof. See case (2) in the proof of Theorem 2 above. □

If $\omega(G)\geqslant 4$ and $b<\omega(G)$, the min–max formula does not necessarily hold even if the graph is perfect and even if it is co-bipartite, as shown in Fig. 3. Notice that this graph arises by replication [30] (from a "house" graph $G' = \overline{P_5}$ for which $\overline{\chi}_3(G') = \sigma_3(G') = 2$). Hence, unlike for perfect graphs, replication does not preserve the validity of the min–max equality. Minimal graphs for which the min–max equality does not hold are studied in [25].

In the following, we prove the min–max equality for interval graphs. Let us first discuss the formula in general:

The Min–max formula in general: The *weak duality* (the "easy part" of the min–max equality) holds for arbitrary graphs:

Weak Duality. *For every graph G and every positive integer b,*

$$\min_{\mathscr{B}\in B(G,b)} |\mathscr{B}|\geqslant \max_{U\subseteq V} \sigma_b(G,U). \tag{5}$$

Proof. For any $U\subseteq V$, a b-clique cover of G trivially induces a b-clique cover of $G(U)$ with no more cliques:

$$\begin{aligned}\min_{\mathscr{B}\in B(G,b)} |\mathscr{B}| = \overline{\chi}_b(G)\geqslant\overline{\chi}_b(G(U)) &= \sum_i \overline{\chi}_b(G(C_i(U)))\\ &\geqslant \sum_i \left\lceil\frac{|C_i(U)|}{b}\right\rceil = \sigma_b(G,U).\end{aligned}$$

This completes the proof of (5). □

The equality does not always hold and not even for perfect graphs as shown in Fig. 3. However, the graph in Fig. 3 contains a circuit without chord. The validity of the min–max formula and the existence of a polynomial algorithm for solving (P2) were recently proved for chordal graphs [25].

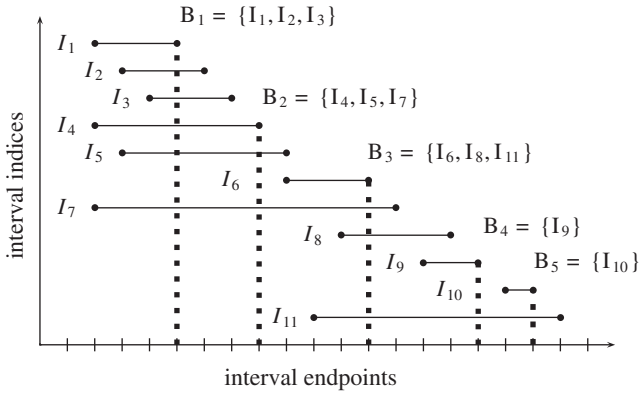

Fig. 4. Optimal solution from GAC for problem (P2) with $b = 3$.

Greedy algorithm and formula for interval graphs: The following algorithm—that we will call GAC (greedy algorithm with compatibility)—constructs a b-clique cover for an interval graph, where the graph is given with an interval representation and a positive integer b is given. In this algorithm, batches (cliques) are successively created in the order $B_1, \ldots, B_\ell$, (ℓ is a positive integer). At any step of the algorithm, a batch B_i is called *unsaturated* if it contains fewer than b intervals, that is, if $1 \leqslant |B_i| < b$; else $|B_i| = b$ and batch B_i is *saturated*.

Consider the tasks in nondecreasing order $I_1, \ldots, I_n$ of their terminal endpoints b_i, breaking ties arbitrarily. The terms "first", "last", "before", "after" will refer to this order. Note, however, that interval J may be after interval I and yet be *placed* before I because of incompatibility of I. The following algorithm finishes each batch before starting another:

The Algorithm GAC. *Construct one batch per iteration until all intervals are placed into batches. In iteration i ($i = 1, \ldots, l$) open a new batch B_i and label it with the first interval $I_j = [a_j, b_j]$ that has not yet been placed in a batch. Starting with I_j place into B_i the first b not yet affected intervals containing b_j (or all of them if they are fewer than b).*

The terminal endpoints of the labels are clearly nondecreasing.

It is easy to see that the following algorithm provides the same result—and shows that GAC can be viewed as a version of the classical "first-fit" algorithm for bin packing (see e.g. [22]), modified to take into account the compatibility constraints: start with no batch and insert interval I_j ($j = 1, \ldots, n$) in the unsaturated batch B_i which has lowest index i and is compatible with I_j; if there is no such (unsaturated and compatible) batch then a new batch is created and interval I_j is put into it.

GAC is illustrated in Fig. 4.

Theorem 4. *For every interval graph G and every positive integer b, GAC solves problem* (P2).

The reformulated version of the algorithm (first-fit with compatibility) stops after $O(n \log n)$ time.

Proof. We proceed by induction of the number of intervals. Let $B_1, B_2, \ldots, B_l$ be a batch sequence constructed by algorithm GAC. Since for the graph $G - B_1$ the sequence $B_2, \ldots, B_l$ is selected by GAC (compare with the definition of the algorithm), it is sufficient to show that *there exists $\mathscr{B} \in B(G, b)$ minimizing $|\mathscr{B}|$ such that $B_1 \in \mathscr{B}$.*

Let the label (the first interval) of B_1 be $I_1 = [a_1, b_1]$ (b_1 is the first endpoint among all). Recall that I_1 is a simplicial vertex. Let D_1 be the batch containing I_1 in an optimal batch sequence $\mathscr{D}$, and suppose $|D_1 \cap B_1|$ is maximum among all possible choices of $\mathscr{D}$. We show $D_1 = B_1$. Note that $I_1 \in D_1$ implies that all intervals in D_1 contain b_1. (They start no later than b_1 by compatibility, and cannot end before b_1, since b_1 is the first terminal endpoint.)

If $|B_1| < b$, then B_1 consists of all neighbors of I_1. Since D_1 is a clique containing I_1, $D_1 \subseteq B_1$. We have then by monotonicity

$$|\mathscr{B}| - 1 = \overline{\chi}_b(G - B_1) \leq \overline{\chi}_b(G - D_1) = |\mathscr{D}| - 1,$$

proving the optimality of $\mathscr{B}$.

Otherwise $|B_1| = b$. Suppose for a contradiction that $I_j \in B_1 \backslash D_1$. Then—since again, $D_1 \subseteq B_1$ can be excluded by monotonicity—there exists $I_k = [a_k, b_k] \in D_1 \backslash B_1$. Recall that B_1 consists of the first b intervals, so $j < k$, $b_j \leqslant b_k$; that $b_1 \in I_k$, as $I_k \in D_1$. Define then $D'_1 := (D_1 \backslash \{I_k\}) \cup \{I_j\}$ and redefine the batch $D \in \mathscr{D}$ containing I_j as $D' := (D \backslash I_j) \cup I_k$. The redefined batches satisfy the compatibility constraint: D'_1 does, since $b_1 \in I_j$, and b_1 is contained in all intervals of D_1 as well; D' does, since I_k meets all intervals met by I_j ($b_j \leqslant b_k$, $b_1 \in I_k$, and all terminal endpoints are greater than or equal to b_1).

On the other hand, $|D'_1 \cap B_1| > |D_1 \cap B_1|$ contradicting the choice of $\mathscr{D}$, and finishing the proof. □

Theorem 5. *For every interval graph G and every positive integer b the following min–max relation holds*:

$$\min_{\mathscr{B} \in B(G,b)} |\mathscr{B}| = \max_{U \subseteq V} \sigma_b(G, U). \tag{6}$$

Proof. By weak duality (5), we know that $\overline{\chi}_b(G) \geqslant \sigma_b(G)$. We prove the reverse inequality by induction on n:

It is obvious for $n = 1$. Suppose it has been proved for graphs with fewer vertices than G. We can suppose that G is connected, otherwise we proceed by components. First note that in case there exists $v \in V(G)$ with $\overline{\chi}_b(G - v) = \overline{\chi}_b(G)$ we are done, since then by the induction hypothesis we have

$$\overline{\chi}_b(G) = \overline{\chi}_b(G - v) = \min_{\mathscr{B} \in B(G-v,b)} |\mathscr{B}| = \min_{U \subseteq V \backslash \{v\}} \sigma_b(G, U) \leqslant \min_{U \subseteq V} \sigma_b(G, U).$$

We can therefore suppose that $\overline{\chi}_b(G - v) < \overline{\chi}_b(G)$ for all $v \in V(G)$. Under this assumption, we show that the batches found by GAC *partition $V(G)$ into cliques of size b and a single batch B with $|B| = 1$*. Indeed, the algorithm outputs at least one unsaturated batch since otherwise $\overline{\chi}_b(G - v) = \overline{\chi}_b(G)$ for any $v \in V(G)$. Let $B \in \mathscr{B}$ be the first unsaturated batch. If $|B| \geqslant 2$ then let L be the last placed interval of B, and let I be the label of B (that is, the first placed interval). Since all intervals placed after L are disjoint from I, the algorithm determines the same batches on $G - L$ as on G (except that B will have one fewer interval), and by Theorem 4 it determines an optimal solution. Therefore $\overline{\chi}_b(G - L) = \overline{\chi}_b(G)$, a contradiction.

So $|B| = 1$. Let t be the terminal endpoint of the unique interval $I \in B$. First, note that the intervals placed after I have their terminal endpoints greater than t; since they do not contain t (otherwise they would be placed in B) their starting point is also greater than t.

We show that the terminal endpoints of intervals placed before I are at most t, implying that such intervals are disjoint from those placed after I. Indeed, if not, let J be the last interval placed before I that contains t. Then $\overline{\chi}_b(G - J) = \overline{\chi}_b(G)$, because none of the intervals placed into the batches strictly after the batch of J until I (including I) can be placed in the batch of J: if any of them would, then they would have been placed there by GAC, not J. The contradiction $\overline{\chi}_b(G - J) = \overline{\chi}_b(G)$ leads to the conclusion that the intervals placed before I are disjoint from those placed after.

Therefore the intervals placed before I form a component of G, and since G is connected this is all G. Then G is clearly of the claimed form and the theorem is proved. □

The proof is algorithmic after extracting the following observations: intervals in unsaturated batches which are not labels of their batch, can be deleted without changing $\overline{\chi}_b$; furthermore, any interval that contains the terminal endpoint of the label of an unsaturated batch can be deleted even if it is in another (possibly saturated) batch. To summarize, we can say that all nonlabel intervals that contain the terminal endpoint of the label of any unsaturated batch can be deleted without decreasing $\overline{\chi}_b$. (The proof actually establishes and uses only the nonexistence of such intervals [containing the label of an unsaturated batch, and not equal to this label] in a minimal counterexample, instead of making use of the stronger statement $\overline{\chi}_b(G - v) < \overline{\chi}_b(G)$.) In the course of the deletion process some batches that were saturated can become unsaturated.

It follows that the following algorithm finds an optimal U for the right-hand side of Theorem 5: browse through intervals in the reverse order $\{I_n, \ldots, I_1\}$ and delete those that contain the terminal endpoints of the label of an unsaturated batch. Some new batches may become unsaturated in this way, and with the exception of the label, all intervals in such batches will in turn be deleted. According to the proof, what remains is an optimal U, more precisely $U \subseteq V(G)$ with the property that all the components of $G(U)$ are partitioned into cliques of size b and a batch of size 1, and in addition $\overline{\chi}_b(G(U)) = \overline{\chi}_b(G)$.

Let us show how this algorithm determines the maximum in Theorem 5 on the example of Fig. 4: I_{11} is deleted since it intersects the unsaturated batch B_5, I_{10} and I_9 are kept since they are labels; I_8 is deleted since it contains the terminal point of the label of the now unsaturated batch $\{I_6, I_8\}$. Finally, the maximum (=5) on the right-hand side in Theorem 5 is attained with $U = \{I_1, I_2, I_3, I_4, I_6, I_9, I_{10}\}$.

(P2) is solvable in polynomial time in split graphs [1,3,16], and can in fact be shown to have the same complexity as bipartite matching [16]. Since split graphs are chordal one cannot expect to extend a greedy type algorithm to chordal graphs for (P2), (unless one finds a greedy algorithm for matching problems). In fact polynomiality of (P2) and Theorem 5 can be proved in chordal graphs using a canonical simplicial decomposition associated with a matching technique [25].

5.3. Problem (P3)

This problem has been posed at the MAPSP 2003 conference and also to visitors in Grenoble. In addition to the authors, two other groups have come up independently in November 2003 with the same polynomial time dynamic programming approach [2,12]. See [22] for references on dynamic programming. (P3) is solvable by a greedy algorithm in P_4-free graphs [11]. On the other hand, (P3) is strongly $\mathcal{NP}$-hard in split graphs and in the complementary of bipartite graphs [11]. Although $\mathcal{NP}$-hard, (P3) is 4-approximable in perfect graphs [28] and within better ratios in some subclasses of perfect graphs, as surveyed in [14]. (P3) is usually studied in its coloring version (equivalent by considering the complementary graph) which is often called *max-coloring*.

Theorem 6. *We can use dynamic programming to solve problem* (P3) *in interval graphs in* $O(n^3)$ *time.*

Proof. Let $\{I_i = [a_i, b_i]\}_{i=1,\ldots,n}$ be a set of intervals on the real line representing graph G. We consider the set X of *endpoints* of the intervals.

$$\begin{aligned} X &= \{a_i\}_{i=1,\ldots,n} \cup \{b_i\}_{i=1,\ldots,n} \cup \{-\infty\} \cup \{+\infty\} \\ &= \{x_1, \ldots, x_q\} \quad \text{with } x_1 < x_2 < \cdots < x_q. \end{aligned}$$

For every pair of values $x_i < x_j \in X$, let $F(x_i, x_j)$ denote the optimum value of the objective function of (P3) for the restricted instance $\mathcal{I}(x_i, x_j)$ (or *subproblem*) consisting of all intervals completely contained in the open interval $]x_i, x_j[$, with $F(x_i, x_j) = 0$ if $\mathcal{I}(x_i, x_j) = \emptyset$. Our dynamic programming approach is based on Lemma 1 below, which implies that we can separate the problem restricted to $\mathcal{I}(x_i, x_j)$ in two subproblems, using an interval of maximum weight. For this, let

$$v(i, j) \in \operatorname{argmax}\{p_a : I_a \in \mathcal{I}(x_i, x_j)\}. \tag{7}$$

Recall that a *maximal clique* is maximal for inclusion, not necessarily for cardinality.

Lemma 1. *There is an optimal schedule for* $\mathcal{I}(x_i, x_j)$ *in which the batch containing* $I_{v(i,j)}$ *is a maximal clique of* $G(\mathcal{I}(x_i, x_j))$.

Proof. Let $S = \{B_1, B_2, \ldots, B_l\}$ be a feasible schedule. W.l.o.g, let $I_{v(i,j)} \in B_1$. Let K be a maximal clique containing B_1. Then the schedule $S' = \{K, B_2 \backslash K, \ldots, B_l \backslash K\}$ is feasible and no worse than S. This is because every batch except the first one has been decreased; batches B_1 and K have the same processing time $p_{v(i,j)}$ because of the choice of $v(i, j)$. □

In light of Lemma 1 one has,

Lemma 2. *For arbitrary fixed $x_i < x_j$ in X, let $v(i, j)$ be defined as in* (7) *and $Y = X \cap I_{v(i,j)}$. The following recursion holds*:

$$F(x_i, x_j) = p_{v(i,j)} + \min_{z \in Y}(F(x_i, z) + F(z, x_j)). \tag{8}$$

Proof. By Lemma 1, it is optimal to include the interval $I_{v(i,j)}$ into some maximal clique B^*. The cost of B^* is $p_{v(i,j)}$. Since B^* is maximal, there is a point z^* in the intersection of all intervals $I_k \in B^*$ such that B^* is the set of all intervals $I_k = [a_k, b_k] \in \mathcal{I}(x_i, x_j)$ satisfying $a_k \leqslant z^* \leqslant b_k$. Thus we may choose z^* in X, and therefore in Y. Given such point z^*, the graph $G(\mathcal{I}(x_i, x_j) \backslash B^*)$ decomposes into two disconnected subgraphs $G(\mathcal{I}(x_i, z^*)$ and $G(\mathcal{I}(z^*, x_j))$ since every interval in $\mathcal{I}(x_i, z^*)$ has its terminal endpoint before the initial endpoint of every interval in $\mathcal{I}(z^*, x_j)$. One can therefore solve the problems on these two subgraphs independently. (See the illustration in Figs. 5–7.) □

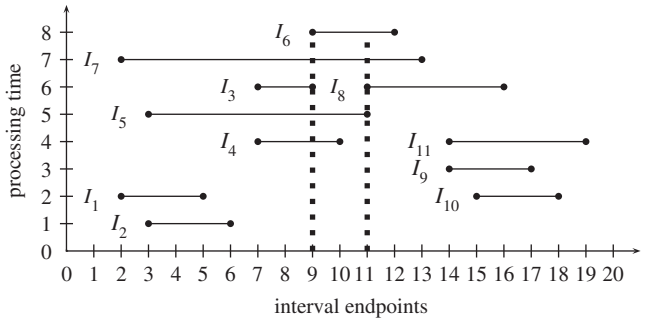

Fig. 5. DP recursion (8) for an instance of (P3). The dashed lines indicate values of z defining the two maximal cliques containing $I_6 = I_{v(-\infty,+\infty)}$, that is, $Y = \{9, 11\}$. Figs. 6 and 7 show the subproblems to be solved for each such maximal clique. The optimum values of the corresponding subproblems are obtained at earlier stages of the DP algorithm.

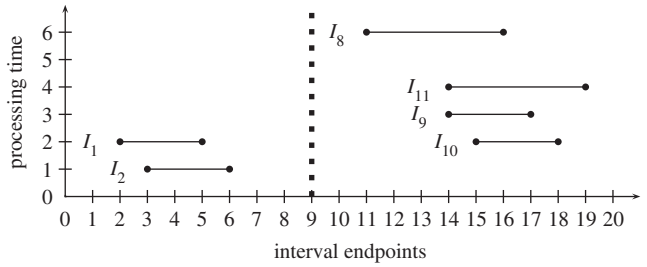

Fig. 6. The two subproblems $\mathcal{I}(1, 9)$ and $\mathcal{I}(9, 20)$ if we choose $z = 9$ (in which case we had $B_1 = \{I_6, I_7, I_3, I_5, I_4\}$).

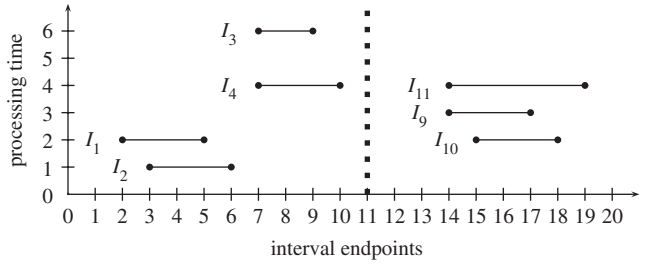

Fig. 7. The two subproblems $\mathcal{I}(1, 11)$ and $\mathcal{I}(11, 20)$ if we choose $z = 11$ (in which case we had $B_1 = \{I_6, I_7, I_8, I_5\}$).

Our dynamic programming algorithm starts from the initial conditions

$$F(x_i, x_{i+1}) = 0 \quad \text{for all } i = 1, \ldots, q-1.$$

Applying the recursion (8) with increasing subproblem width $x_j - x_i$, it computes an optimal schedule

$$S(x_i, x_j) = \begin{cases} \emptyset & \text{if } \mathscr{I}(x_i, x_j) = \emptyset, \\ S(x_i, z^*) \cup B^* \cup S(z^*, x_j) & \text{otherwise.} \end{cases}$$

The optimum value is $C_{\max} = F(x_1, x_q)$, and $S(x_1, x_q)$ is an optimal solution. Since there are $O(q^2) = O(n^2)$ subproblems and $O(q) = O(n)$ candidate values for z in each subproblem, the resulting dynamic programming algorithm solves problem (P3) in $O(n^3)$ time. This completes the proof of Theorem 6. □

This approach extends to optimal partition of circular-arc graphs into Helly-cliques.[1] The following algorithm provides an optimal partition into Helly-cliques in time $O(n^3)$. Extend the definition of $F(x_i, x_j)$ to any pair (x_i, x_j) of distinct endpoints, by considering that x_i is before x_j in clockwise order on the circle. Since the resulting subproblem $\mathscr{I}(x_i, x_j)$ reduces to that on an (ordinary) interval graph, recursion (8) still applies and yields all $O(n^2)$ values $F(x_i, x_j)$ in $O(n^3)$ time. Now, there is an optimal partition into Helly-cliques in which the batch containing $\mathscr{I}(x_i, x_j)$ (defined by Eq. (7)) is a maximal Helly-clique of G. Indeed, the proof of Lemma 1 extends to optimal partitions into edges of a hereditary hypergraph [25]. Finally, we only need to choose the best of the $O(n)$ values $F(x_i, x_j)$ where x_j immediately precedes x_i in clockwise order and both endpoints are contained in interval $I_{v(i,j)}$. This gives a $O(n^3 + n) = O(n^3)$ algorithm for the optimum partition into Helly-cliques of circular-arc graphs.

5.4. Problem (P4)

For general graphs, the problem

$$1/p\text{-}batch, G = (V, E), \quad b = 2/C_{\max}$$

is solvable in polynomial time [7]. Indeed, an optimal solution can be obtained by transforming the problem into a maximum weight matching problem as follows. Assign the weights $\min\{p_i, p_j\} = p_i + p_j - \max\{p_i, p_j\}$ to the edges $(i, j) \in E$ and solve the maximum weight matching problem in G. Then for each edge of the matching, process the corresponding two tasks in the same batch. The other tasks are processed as single task batches.

Boudhar [3] shows that problem

$$1/p\text{-}batch, G = (V, E), \quad b = k/C_{\max}$$

is $\mathscr{NP}$-hard for split graphs, for every $k \geqslant 3$. The complexity status of problem (P4) for interval graphs and $k \geqslant 3$ remains open.

6. Batching with release dates and other extensions

So far, we have assumed that all tasks are available at the same time. This is appropriate for Application II. However, in Application I, one typically has several successive cycles of rolling and heating so that there is a flow of the material to the furnace, resulting in different release dates r_j for the tasks T_j. Hence, we define problems (P′1)–(P′4) by adding release dates r_j to the corresponding problems (P1)–(P4). Application I would then be modelled by problems (P′2) and (P′4).

Batching with arrival times changes the nature of the problem substantially, as it is no longer sufficient to just partition the tasks into batches: we must also schedule these batches to take into account the release dates. Defining the earliest date $r(B) = \max\{r_j : T_j \in B\}$, at which a batch B can be started, the batch completion times $c(B_i)$ of a

[1] This does not provide a solution for problem (P3) in this class of graphs. However, cyclic scheduling problems have a more natural link with partitions into Helly-cliques than with partitions into cliques.

feasible batch schedule must now satisfy the *release date constraints* $c(B_i) \geqslant r(B_i) + p(B_i)$. The schedule makespan is $C_{\max} = \max_i c(B_i)$. Boudhar [5] shows that problem (P′2) with unit processing times and capacity $b = 2$,

$$1/p\text{-}batch, G = (V, E), \quad b = 2, r_i, \quad p_i = 1/C_{\max}$$

is strongly $\mathcal{NP}$-hard for split graphs. The complexity status of this problem for interval graphs is unknown.

Further extensions, arising in connection with Application I, may be defined. On one hand, it may be more appropriate to replace $C_{\max}$ with a flow time criterion, since one would like to reduce the storage of the semi-finished products on the shop floor. On the other hand, the compatibility relations may also be extended. Rather than only considering compatibility with respect to the height of the metal coils, one might also add compatibility for the diameters, weights, etc. Each characteristic (measurement) has its tolerance limits and therefore defines an interval graph. This leads to a compatibility graph which is the intersection of interval graphs. Such a graph may neither be an interval graph, nor even a perfect graph.

Acknowledgment

This research is supported by INTAS Grant 03-51-5501 as well as by the ADONET network of the European Community, which is a Marie Curie Training Network. We thank our colleagues Claudson Bornstein, Jayme Szwarcfiter, Emmanuel Desgrippes and Christophe Rapine for sharing with us their results [2,12], Mourad Boudhar for pointing out his results in Ref. [3] and Bruno Escoffier for sharing his knowledge concerning max-coloring [14].

References

[1] H.L. Bodlaender, K. Jansen, Restrictions of graph partition problems, part I, Theoret. Comput. Sci. 148 (1) (1995) 93–109.
[2] C. Bornstein, J.L. Szwarcfiter, Personal communication, Rio de Janeiro, November 2003.
[3] M. Boudhar, Static scheduling on a single batch processing machine with split compatibility graphs, Cahier No. 28, Laboratoire Leibniz, Grenoble, 2001.
[4] M. Boudhar, Scheduling a batch processing machine with bipartite compatibility graphs, Math. Methods Oper. Res. 57 (2003) 327–513.
[5] M. Boudhar, Dynamic scheduling on a single batch processing machine with split compatibility graphs, J. Math. Modelling Algorithms 2 (2003) 17–35.
[6] M. Boudhar, G. Finke, Scheduling on batch processing machines with constraints of compatibility between jobs, in: Proceedings of the Second Conference on Management and Control of Production and Logistics (MCPL'2000), vol. 2, Grenoble, 2000, pp. 703–708.
[7] M. Boudhar, G. Finke, Scheduling on a batch machine with job compatibilities, Belgian J. Oper. Res. Statist. Comput. Sci. 40 (2000) 69–80.
[8] N. Brauner, C. Dhaenens-Flipo, M.-L. Espinouse, G. Finke, H. Gavranovic, Decomposition into parallel work phases with application to the sheet metal industry, in: Proceedings of the International Conference on Industrial Engineering and Production Management (IEPM'99), vol. 1, Glasgow, 1999, pp. 389–396.
[9] P. Brucker, A. Gladky, H. Hoogeveen, M.Y. Kovalyov, C. Potts, T. Tautenhahn, S. van de Velde, Scheduling a batching machine, J. Scheduling 1 (1998) 31–54.
[10] P. Brucker, S. Knust, Complexity results of scheduling problems, ⟨www.mathematik.uni-osnabrueck.de/research/OR/class/⟩.
[11] M. Demange, D. de Werra, J. Monnot, V.T. Paschos, Time slot scheduling of compatible jobs, Cahier du Lamsade No. 182, Université Paris IX, Dauphine, Paris, 2001.
[12] E. Desgrippes, C. Rapine, Personal communication, Grenoble, November 2003.
[14] B. Escoffier, Approximation polynomiale de problèmes d'optimisation: Aspects structurels et opérationnels, Ph.D. Thesis, Laboratoire LAMSADE, Paris Dauphine, 2005.
[15] M.R. Garey, D.S. Johnson, Computers and Intractability, Freeman, San Francisco, CA, 1979.
[16] F. Gardi, Ordonnancement avec exclusion mutuelle par un graphe d'intervalle ou une classe apparentée: complexité et algorithmes, Ph.D. Thesis, Laboratoire d'informatique fondamentale, Faculté des sciences de Luminy, 2005.
[17] H. Gavranovic, Affectation de fréquences et conception optimale d'une ligne de production: modèles et algorithmes de résolution, Ph.D. Thesis, Université Joseph Fourier, Grenoble, 2002.
[18] H. Gavranovic, G. Finke, Graph partitioning and set covering for the optimal design of a production system in the metal industry, in: Proceedings of the Second Conference on Management and Control of Production and Logistics (MCPL'2000), vol. 2, Grenoble, 2000, pp. 603–608.
[19] M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, New York, 1980.
[20] M. Grötschel, L. Lovász, A. Schrijver, Geometric Algorithms and Combinatorial Optimization, Springer, Berlin, 1988.
[21] P. Hansen, A. Hertz, J. Kuplinsky, Bounded vertex colorings of graphs, Discrete Math. 111 (1993) 305–312.
[22] D.S. Hochbaum, Approximation Algorithms for $\mathcal{NP}$-Hard Problems, PWS Publishing Company, 1995.
[23] D.S. Hochbaum, D. Landy, Scheduling semiconductor burn-in operations to minimize total flowtime, Oper. Res. 45 (1997) 874–885.

[24] K. Jansen, The mutual exclusion scheduling problem for permutation and comparability graphs, Inform. and Comput. 180 (2) (2003) 71–81.
[25] V. Jost, Ordonnancement chromatique: Polyédres, Complexité et Classification, Ph.D. Thesis, Laboratoire Leibniz, UJF, Grenoble, 2006.
[26] P.L. Nguyen, Planification tactique de la production: approche hiérarchisée pour une classe d'entreprises de sous-traitance et application au cas de laminage à froid, Ph.D. Thesis, Université Joseph Fourier, Grenoble, 1997.
[27] C.H. Papadimitriou, M. Yannakakis, Scheduling interval-ordered tasks, SIAM J. Comput. 8 (1979) 405–409.
[28] S.V. Pemmaraju, R. Ramman, Approximation algorithms for the max-coloring problem, ICALP'05, 2005, accepted for publication.
[29] C.N. Potts, M.Y. Kovalyov, Scheduling with batching: a review, European J. Oper. Res. 120 (2000) 228–240.
[30] A. Schrijver, Combinatorial Optimization: Polyhedra and Efficiency, Springer, Berlin, 2003.